# Contents

# 1. Introduction:

Digital filters are mathematical algorithms that operate on discrete-time signals, transforming them by attenuating or enhancing certain frequency components. They are widely used in various applications, including audio processing, image processing, data transmission, and biomedical signal processing. Unlike analog filters, digital filters offer flexibility, programmability, and scalability, making them more adaptable to changing requirements. They can be designed using various techniques, such as finite impulse response (FIR) and infinite impulse response (IIR) filters.

Before analog signals undergo further processing, they need to traverse a physical component known as analog filtering. The analog data is then transmitted to a processor, where the software takes charge of digital filtering. The primary role of filters is to completely block certain frequencies while permitting others to pass through without changes. Frequencies allowed to pass constitute the passband, while those obstructed belong to the stopband. The transitional area in between is known as the transitional band. A quick roll-off indicates a relatively narrow transitional band. The cutoff frequency serves as the demarcation between the passband and the transitional band.

The four main categories of filters are the low-pass filter, high-pass filter, band-pass filter, and band-stop filter (also known as band-reject or notch filter). A digital filter employs a digital processor to conduct numerical computations on sampled signal values. This processor can be a general-purpose computer like a PC or a dedicated DSP (Digital Signal Processor) chip.

The Low-pass filters allow low-frequency signals to pass through while blocking high-frequency signals as shown in Fig.1.2. Low-pass filters are often used to clean up signals, remove noise, create a smoothing effect, perform data averaging, and design decimators and interpolators.

High-pass filters allow high-frequency signals to pass through while blocking low-frequency signals. They are commonly used in audio and video applications to enhance high-frequency components and remove noise and unwanted low-frequency components.

Band-pass filters allow signals within a specific frequency range to pass through while blocking signals outside that range. They are commonly used in telecommunications and electronics to isolate specific frequency bands for transmission or reception.

Band-stop filters, also known as band-reject filters, block signals within a specific frequency range while allowing signals outside that range to pass through. They are commonly used in telecommunications and electronics to eliminate interference or unwanted signals in a specific frequency band.

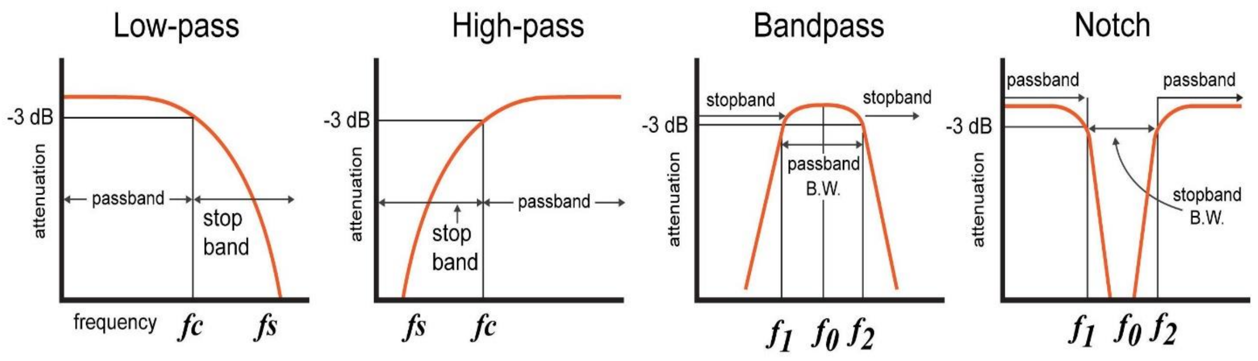


Fig. 1.1 – Frequency Response of Filters

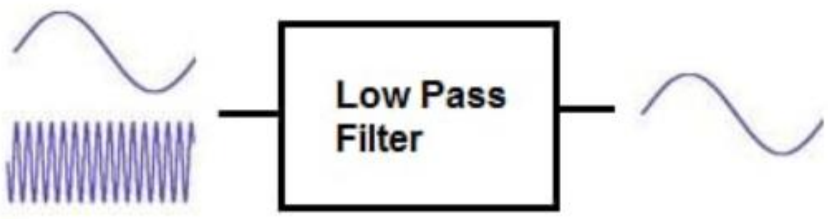


Fig. 1.2

## 2. Objectives:

### 2.1. To design a Low Pass Filter by giving cutoff frequencies.

A low pass filter has been designed by giving the input sample frequency and a cutoff frequency, The Order of the filter, magnitude spectrum, phase spectrum, and pole-zero plot have been obtained as well as its stability nature, hence we can design a filter according to requirements.

### 2.2. To get the results of a low pass filter by eliminating high-frequency components from noisy audio.

An input audio file that has high-frequency values in it is taken. Here we have designed a low pass filter that stops the high frequencies and the output of the signal is clear audio with less noise in it. This can also be observed well by using sound functions to play the input and output audio files.

### 2.3. To compare the graphs of input, and output signals, obtained from time and frequency response.

After the Implementation of each step, the graph is to be observed, at first frequency response of the input signal, noisy signal, and output signal, then the time domain response of the input and output signal, followed by the power spectrum of input signal, noisy signal and output signal in this way a comparison has to be made between these graphs of input, noisy and output results.

# 3. Theoretical Background

Digital Signal Processing (DSP) filters are fundamental components in the field of signal processing, playing a crucial role in modifying or extracting information from signals. These filters are designed to operate on digital signals, which are discrete-time representations of continuous signals. DSP filters are used in a wide range of applications, including telecommunications, audio processing, image processing, biomedical signal processing, and many others.

**Types of Filters:**

**1. Impulse Response (FIR) Filters:**

- FIR filters have a finite impulse response, meaning that their output is solely determined by a weighted sum of the current and past input samples.
- FIR filters are characterized by linear phase response, making them suitable for applications where phase distortion must be minimized.
- The filter coefficients are often referred to as the "taps," and the design of FIR filters involves determining these coefficients to achieve desired frequency response characteristics.

**2. Infinite Impulse Response (IIR) Filters:**

- IIR filters have an infinite impulse response, meaning that their output depends on both current and past input samples, creating feedback loops.
- IIR filters are commonly used when a more compact filter design is desired, but they may exhibit nonlinear phase response.
- Butterworth, Chebyshev, and elliptic filters are examples of IIR filters, each with specific characteristics suited for different applications

**WINDOWING METHOD:**

Windowing is the process of taking a small subset of a larger dataset, for processing and analysis. A naive approach, the rectangular window, involves simply truncating the dataset 7 before and after the window, while not modifying the contents of the window at all. However, as we will see, this is a poor method of windowing and causes power leakage.

**KAISER WINDOW:**

A Kaiser window is a type of window function used in signal processing and spectral analysis. It is defined by a parameter called the beta value, which controls the trade-off between the main lobe width and the side lobe level in the window's frequency response.

**Applications of the Kaiser window include:**

1. Spectral Analysis: It is commonly used in the design of finite impulse response (FIR) filters and spectral analysis techniques like the fast Fourier transform (FFT). The Kaiser window helps to reduce the leakage effect in the frequency domain.
2. Digital Signal Processing (DSP): In DSP applications, the Kaiser window is often employed to shape the frequency response of filters and minimize undesirable effects like spectral leakage.
3. Speech and Audio Processing: Kaiser windows can be applied in speech processing to improve the accuracy of techniques such as windowing and filtering.

In essence, the Kaiser window is versatile and finds applications in various domains where precise frequency analysis and control of spectral characteristics are crucial.

**Working of Kaiser window:**

Imagine you're trying to listen to a song on the radio, but the signal is a bit messy. The Kaiser window helps clean up this signal.

It works by shaping the way we look at chunks of the signal. It's like putting a special window over part of the signal to focus on specific details. The Kaiser window has a parameter (called beta) that lets you control how wide or narrow this window is.

**This control is handy because it helps to balance two things:**

Main Lobe Width: This is like the main part of the window that focuses on the important stuff.

Side Lobes: These are like extra bits on the sides. Too much of them can cause interference or messiness.

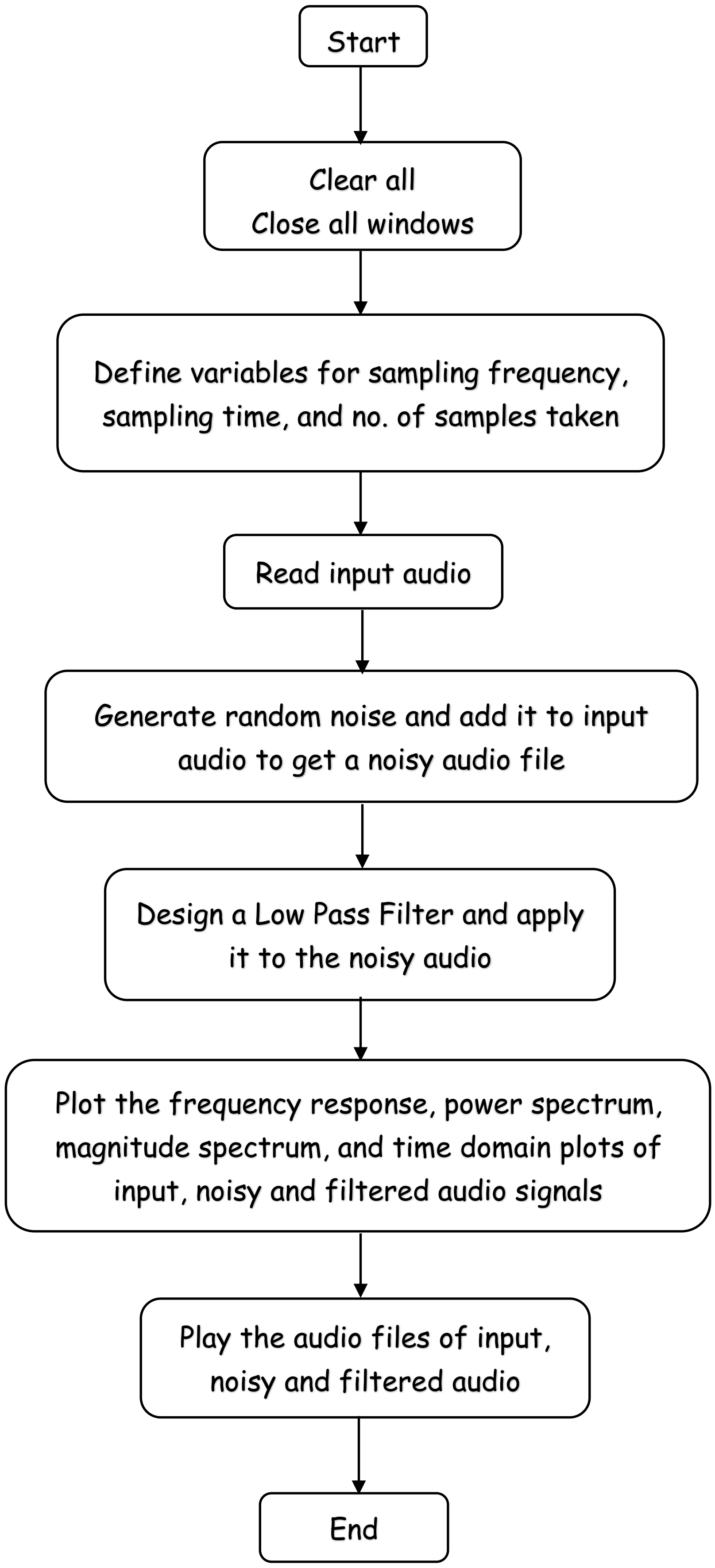
So, by adjusting the beta value, you can decide how much you want to focus on the main part of the signal and how much you want to ignore the extra stuff on the sides. It's a way to get a clearer picture of the signal you're interested in.

**Random Noise:**

Random noise refers to unwanted or unpredictable fluctuations in a signal, data set, or process. It often appears as random variations with no discernible pattern. In various fields, such as signal processing or statistics, managing or minimizing random noise is crucial for obtaining accurate and meaningful results.

Random noise is a significant consideration in digital signal processing (DSP). In DSP, random noise can affect the quality of signals during acquisition, transmission, or processing. Techniques like filtering, smoothing, or error correction algorithms are commonly used to mitigate the impact of random noise on digital signals. Understanding and managing random noise in DSP is crucial for ensuring the accuracy and reliability of digital systems and data.

## 4. Flowchart


Start
Clear all
Close all windows
Define variables for sampling frequency, sampling time, and no. of samples taken
Read input audio
Generate random noise and add it to input audio to get a noisy audio file
Design a Low Pass Filter and apply it to the noisy audio
Plot the frequency response, power spectrum, magnitude spectrum, and time domain plots of input, noisy and filtered audio signals
Play the audio files of input, noisy and filtered audio
End

## 5. Code

```
clearvars
clc
close all

[InAud,Fs] = audioread('InputAudio_V.wav');

L = length(InAud);          %length of the input sequence
T = 1/Fs;
t = (0:L-1)*T;   %time in seconds
A1 = 0.02;       %Amplitude of Random noise
A2 = 0.05;

Tr_InAud = transpose(InAud);
%Noisy audio is created by adding random noise to the main input audio
NoisyAud_1 = Tr_InAud + A1*randn(1,length(Tr_InAud));
NoisyAud_2 = Tr_InAud + A2*randn(1,length(Tr_InAud));

audiowrite("NoisyAudio_1.wav",NoisyAud_1,Fs);
audiowrite("NoisyAudio_2.wav",NoisyAud_2,Fs);


% Coefficients of the designed filter are loaded
load("Num.mat");

%Noisy audio is filtered
FilteredAud_1 = filter(Num,1,NoisyAud_1);
FilteredAud_2 = filter(Num,1,NoisyAud_2);

audiowrite("OutputAudio_1.wav",FilteredAud_1,Fs);
audiowrite("OutputAudio_2.wav",FilteredAud_2,Fs);

Order = filtord(Num);      %order of the filter

%%
%Figures

%Time domain plots
%figure 1
figure
subplot(3,2,1)
plot(t, InAud);
title('Input signal: Time-domain');
xlabel('time(seconds)');
ylabel('Amplitude');
grid on
subplot(3,2,3)
plot(t, NoisyAud_1, 'r');
title('Noisy signal (A=0.02): Time-domain');
xlabel('time(seconds)');
ylabel('Amplitude');
grid on
subplot(3,2,5)
plot(t, FilteredAud_1, 'g');
title("Output signal: Time-domain");
xlabel('time(seconds)');
```

```
ylabel('Amplitude');
grid on

subplot(3,2,2)
plot(t, InAud);
title('Input signal: Time-domain');
xlabel('time(seconds)');
ylabel('Amplitude');
grid on
subplot(3,2,4)
plot(t, NoisyAud_2, 'r');
title('Noisy signal (A=0.05): Time-domain');
xlabel('time(seconds)');
ylabel('Amplitude');
grid on
subplot(3,2,6)
plot(t, FilteredAud_2, 'g');
title("Output signal: Time-domain");
xlabel('time(seconds)');
ylabel('Amplitude');
grid on
%%


%Magnitude spectrum plots
%figure 2

figure
Length_audio = length(InAud);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,1)
FFT_InAud = fft(InAud); %calculating fft of Input audio
mag = abs(FFT_InAud(1:Length_audio/2+1));
plot(f,mag);
title('Magnitude spectrum of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(NoisyAud_1);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,3)
FFT_NoisyAud_1 = fft(NoisyAud_1); %calculating fft of Noisy audio
mag = abs(FFT_NoisyAud_1(1:Length_audio/2+1));
plot(f,mag, 'r');
title('Magnitude spectrum of Noisy Audio 1');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(FilteredAud_1);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,5)
FFT_FilteredAud_1 = fft(FilteredAud_1); %calculating fft of Input audio
mag = abs(FFT_FilteredAud_1(1:Length_audio/2+1));
plot(f,mag, 'g');
title('Magnitude spectrum of Output Audio 1');
xlabel('Frequency(Hz)');
```

```
ylabel('Magnitude');
grid on

Length_audio = length(InAud);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,2)
FFT_InAud = fft(InAud); %calculating fft of Input audio
mag = abs(FFT_InAud(1:Length_audio/2+1));
plot(f,mag);
title('Magnitude spectrum of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(NoisyAud_2);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,4)
FFT_NoisyAud_2 = fft(NoisyAud_2); %calculating fft of Noisy audio
mag = abs(FFT_NoisyAud_2(1:Length_audio/2+1));
plot(f,mag, 'r');
title('Magnitude spectrum of Noisy Audio 2');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(FilteredAud_2);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,6)
FFT_FilteredAud_2 = fft(FilteredAud_2); %calculating fft of Input audio
mag = abs(FFT_FilteredAud_2(1:Length_audio/2+1));
plot(f,mag, 'g');
title('Magnitude spectrum of Output Audio 2');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on



%% Log scale Magnitude Spectrum plots
%figure 3
figure
Length_audio = length(InAud);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,1)
FFT_InAud = fft(InAud); %calculating fft of Input audio
mag = abs(FFT_InAud(1:Length_audio/2+1));
logMag = 20*log10(mag);
plot(f,logMag);
title('Log Scale Magnitude spectrum of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Magnitude (dB)');
grid on

Length_audio = length(NoisyAud_1);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,3)
FFT_NoisyAud_1 = fft(NoisyAud_1); %calculating fft of Noisy audio
mag = abs(FFT_NoisyAud_1(1:Length_audio/2+1));
```

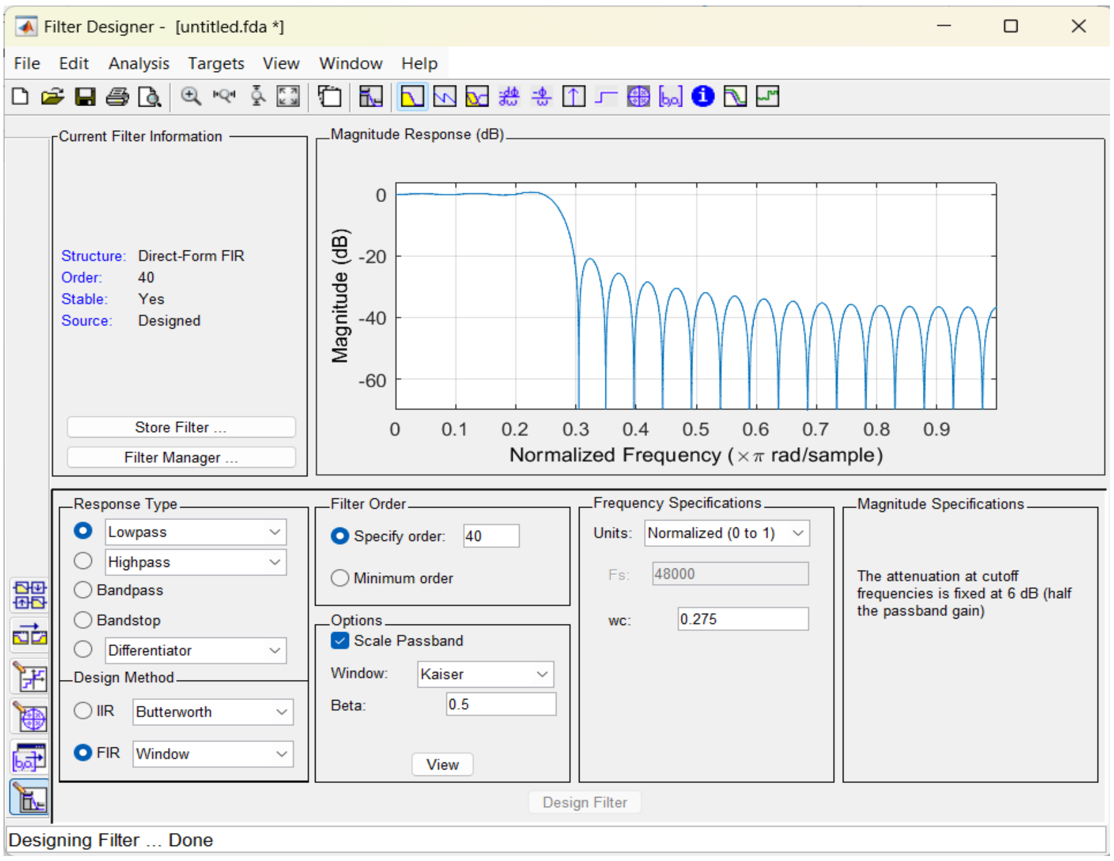
```
logMag = 20*log10(mag);
plot(f,logMag, 'r');
title('Log Scale Magnitude spectrum of Noisy Audio 1');
xlabel('Frequency(Hz)');
ylabel('Magnitude(dB)');
grid on

Length_audio = length(FilteredAud_1);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,5)
FFT_FilteredAud_1 = fft(FilteredAud_1); %calculating fft of Input audio
mag = abs(FFT_FilteredAud_1(1:Length_audio/2+1));
logMag = 20*log10(mag);
plot(f,logMag, 'g');
title('Log Scale Magnitude spectrum of Output Audio 1');
xlabel('Frequency(Hz)');
ylabel('Magnitude (dB)');
grid on

Length_audio = length(InAud);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,2)
FFT_InAud = fft(InAud); %calculating fft of Input audio
mag = abs(FFT_InAud(1:Length_audio/2+1));
logMag = 20*log10(mag);
plot(f,logMag);
title('Log Scale Magnitude spectrum of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Magnitude (dB)');
grid on

Length_audio = length(NoisyAud_2);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,4)
FFT_NoisyAud_2 = fft(NoisyAud_2); %calculating fft of Noisy audio
mag = abs(FFT_NoisyAud_2(1:Length_audio/2+1));
logMag = 20*log10(mag);
plot(f,logMag, 'r');
title('Log Scale Magnitude spectrum of Noisy Audio 2');
xlabel('Frequency(Hz)');
ylabel('Magnitude (dB)');
grid on

Length_audio = length(FilteredAud_2);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,6)
FFT_FilteredAud_2 = fft(FilteredAud_2); %calculating fft of Input audio
mag = abs(FFT_FilteredAud_2(1:Length_audio/2+1));
logMag = 20*log10(mag);
plot(f,logMag, 'g');
title('Log Scale Magnitude spectrum of Output Audio 2');
xlabel('Frequency(Hz)');
ylabel('Magnitude (dB)');
grid on
```

```
%% Power Spectrum Plots
figure
Length_audio = length(InAud);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,1)
FFT_InAud = fft(InAud); %calculating fft of Input audio
mag = abs(FFT_InAud(1:Length_audio/2+1));
power = mag.^2;
plot(f,power);
title('Power spectrum of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(NoisyAud_1);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,3)
FFT_NoisyAud_1 = fft(NoisyAud_1); %calculating fft of Noisy audio
mag = abs(FFT_NoisyAud_1(1:Length_audio/2+1));
power = mag.^2;
plot(f,power, 'r');
title('Power spectrum of Noisy Audio 1');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(FilteredAud_1);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,5)
FFT_FilteredAud_1 = fft(FilteredAud_1); %calculating fft of Input audio
mag = abs(FFT_FilteredAud_1(1:Length_audio/2+1));
power = mag.^2;
plot(f,power, 'g');
title('Power spectrum of Output Audio 1');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(InAud);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,2)
FFT_InAud = fft(InAud); %calculating fft of Input audio
mag = abs(FFT_InAud(1:Length_audio/2+1));
power = mag.^2;
plot(f,power);
title('Power spectrum of Input Audio');
xlabel('Frequency(Hz)');
ylabel('Magnitude');

Length_audio = length(NoisyAud_2);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,4)
FFT_NoisyAud_2 = fft(NoisyAud_2); %calculating fft of Noisy audio
mag = abs(FFT_NoisyAud_2(1:Length_audio/2+1));
plot(f,mag);
power = mag.^2;
plot(f,power, 'r');
title('Power spectrum of Noisy Audio 2');
```

```
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on

Length_audio = length(FilteredAud_2);
f = linspace(0,Fs/2,Length_audio/2+1); %Frequency Spectrum: 0 to fs/2
subplot(3,2,6)
FFT_FilteredAud_2 = fft(FilteredAud_2); %calculating fft of Input audio
mag = abs(FFT_FilteredAud_2(1:Length_audio/2+1));
power = mag.^2;
plot(f,power, 'g');
title('Power spectrum of Output Audio 2');
xlabel('Frequency(Hz)');
ylabel('Magnitude');
grid on


%%
figure
zplane(Num,1);
title('Pole-zero Plot');

figure
freqz(Num,1,L);
title('Frequency Response');

figure
impz(Num,1);
title('Impulse Response');

%%
%Playing Audio
sound(InAud,Fs)
pause

sound(NoisyAud_1,Fs)
pause

sound(FilteredAud_1,Fs)
pause

sound(NoisyAud_2,Fs)
pause

sound(FilteredAud_2,Fs)
```

## 6. Result and Future Scope

The normalized low pass FIR filter is designed with a cut-off frequency of 0.275π (2200 Hz) using a Kaiser window with beta 0.5 and its order is 40.



Structure: Direct-Form FIR
Order: 40
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR] Window

Filter Order: [✓ Specify order: 40] Minimum order

Options: [✓ Scale Passband]
Window: Kaiser
Beta: 0.5

Units: Normalized (0 to 1)
Fs: 48000
wc: 0.275

The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

Designing Filter ... Done

Fig. 6.1: FDA tool of MATLAB used for filter designing

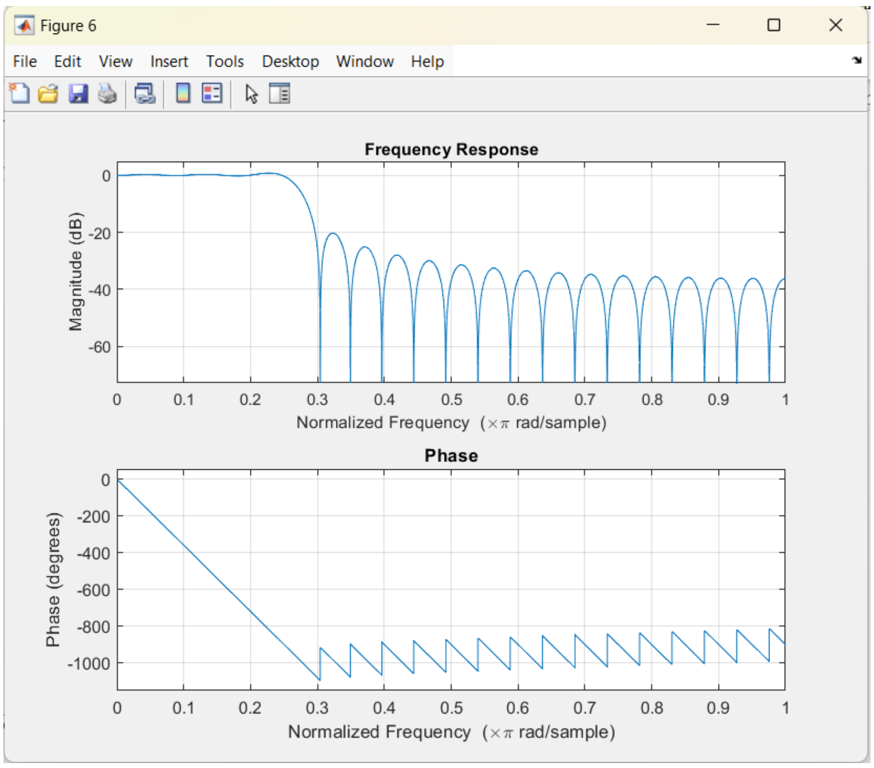
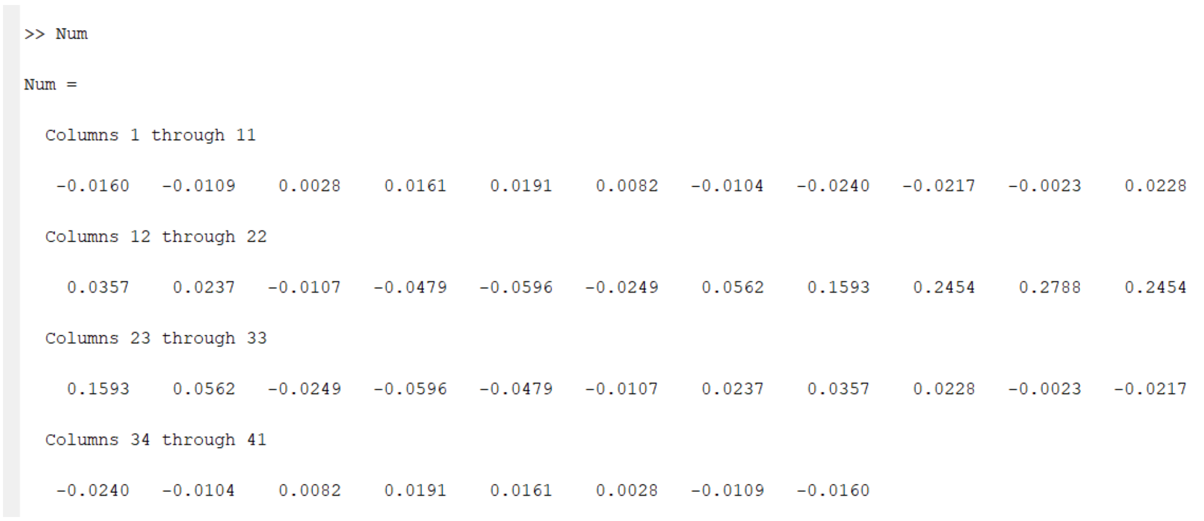


Fig. 6.2: Magnitude and Phase response of the designed filter



```
>> Num

Num =

  Columns 1 through 11

   -0.0160   -0.0109    0.0028    0.0161    0.0191    0.0082   -0.0104   -0.0240   -0.0217   -0.0023    0.0228

  Columns 12 through 22

    0.0357    0.0237   -0.0107   -0.0479   -0.0596   -0.0249    0.0562    0.1593    0.2454    0.2788    0.2454

  Columns 23 through 33

    0.1593    0.0562   -0.0249   -0.0596   -0.0479   -0.0107    0.0237    0.0357    0.0228   -0.0023   -0.0217

  Columns 34 through 41

   -0.0240   -0.0104    0.0082    0.0191    0.0161    0.0028   -0.0109   -0.0160
```

Fig. 6.3: Coefficients of the designed filter

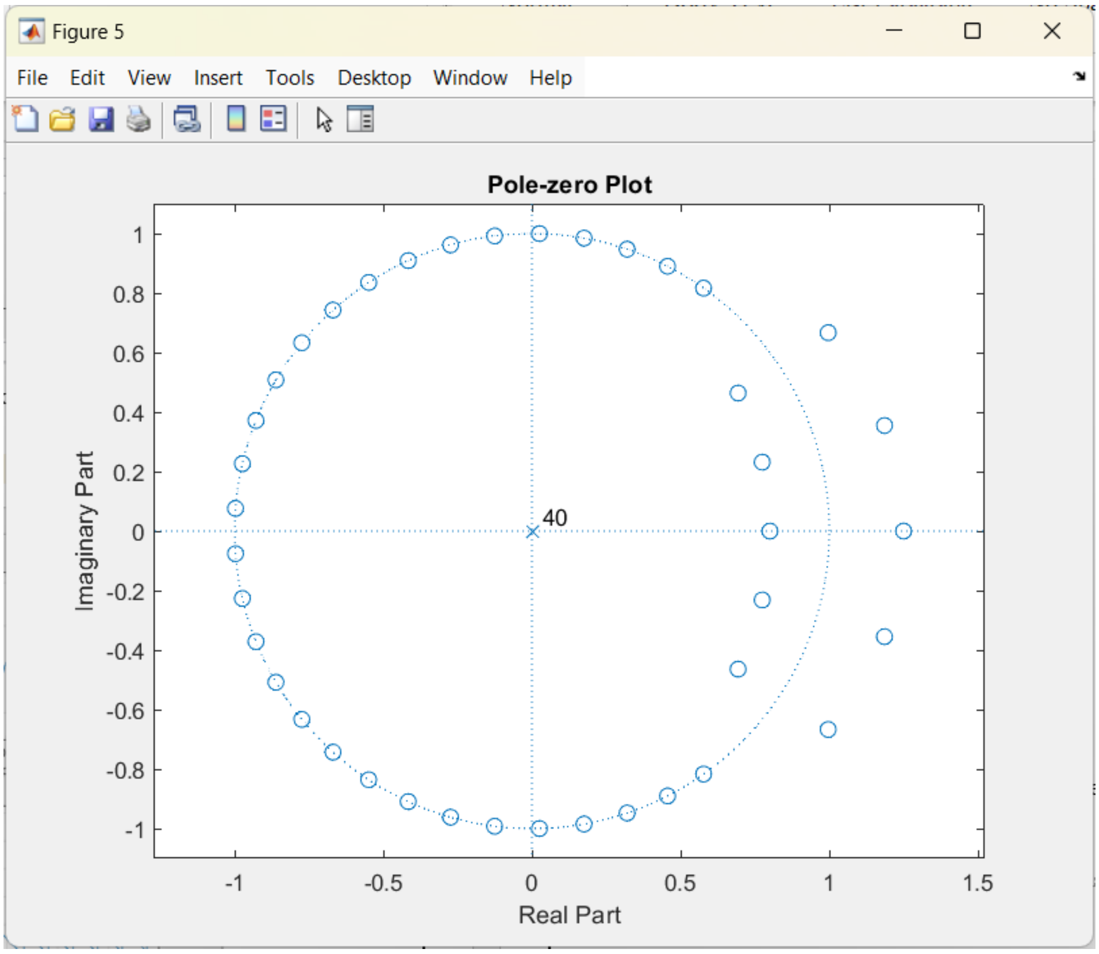


Fig. 6.4: Pole-zero plot

We can see that there are no poles and only zeroes are present.

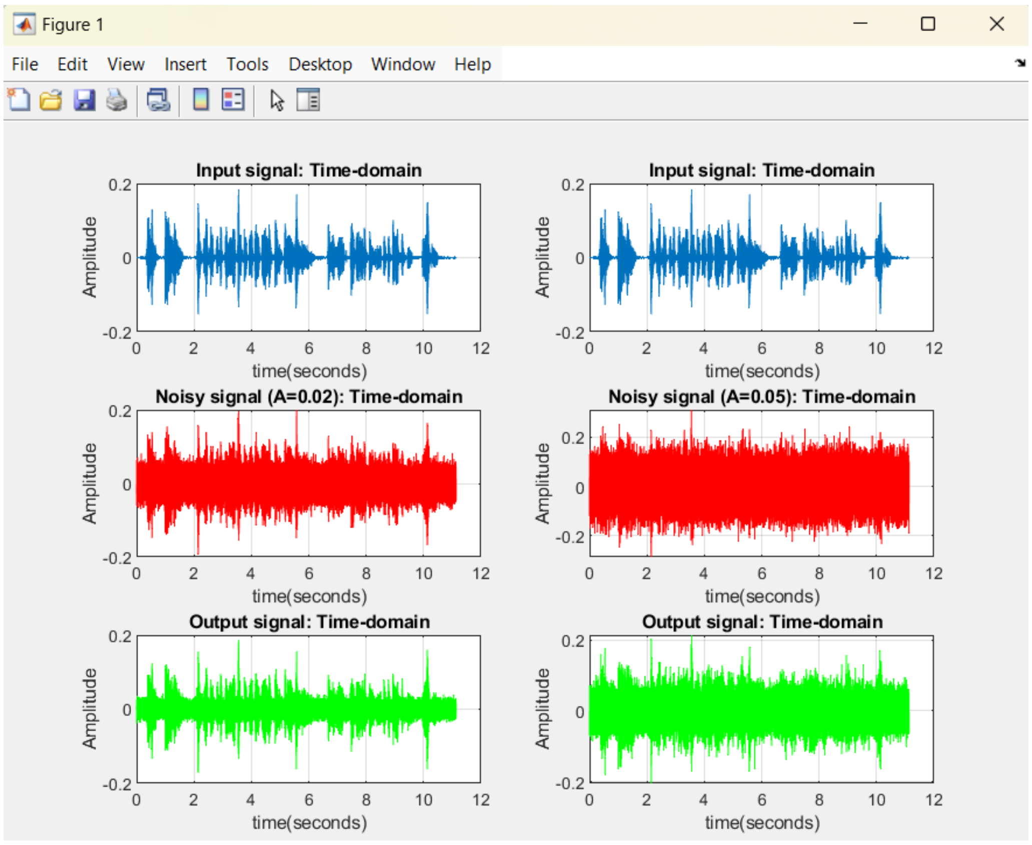


Fig. 6.5: Time domain plots of input audio, noisy audio, and output audio

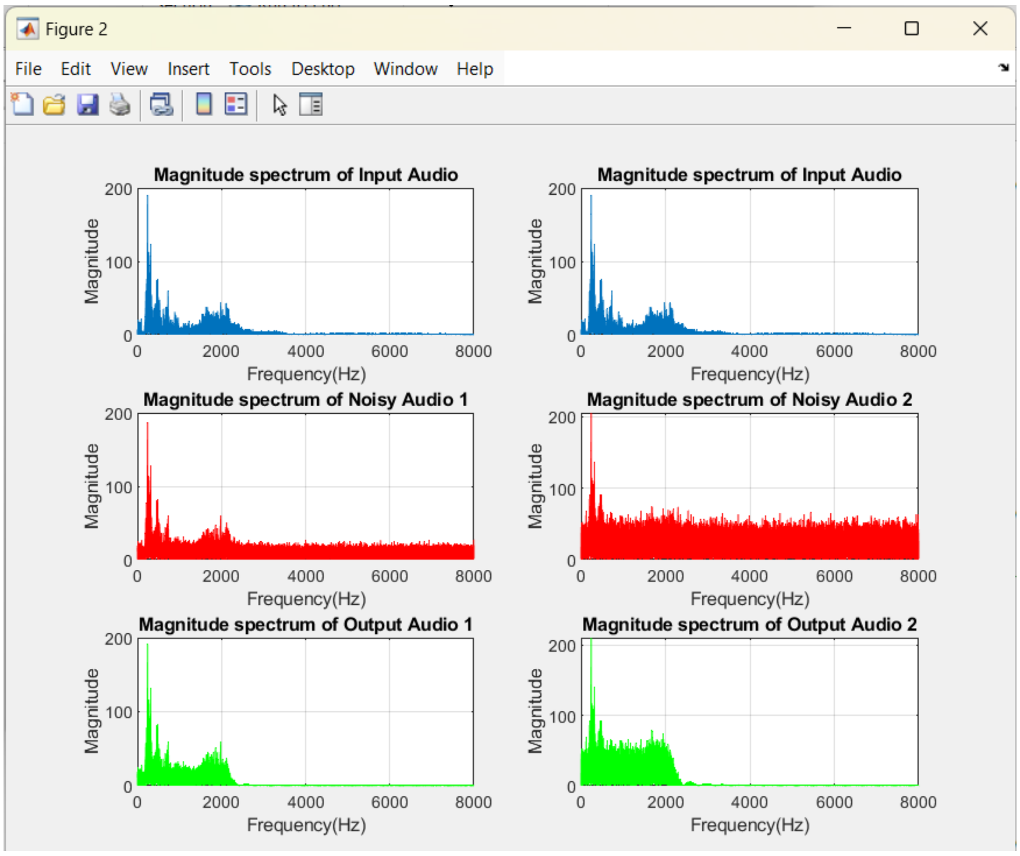


Fig. 6.6: Magnitude spectrum of Input, Two noisy signals with amplitude 0.02 and 0.05, and the respective output signals after filtering.

This shows that frequencies higher than the cutoff frequency are removed.

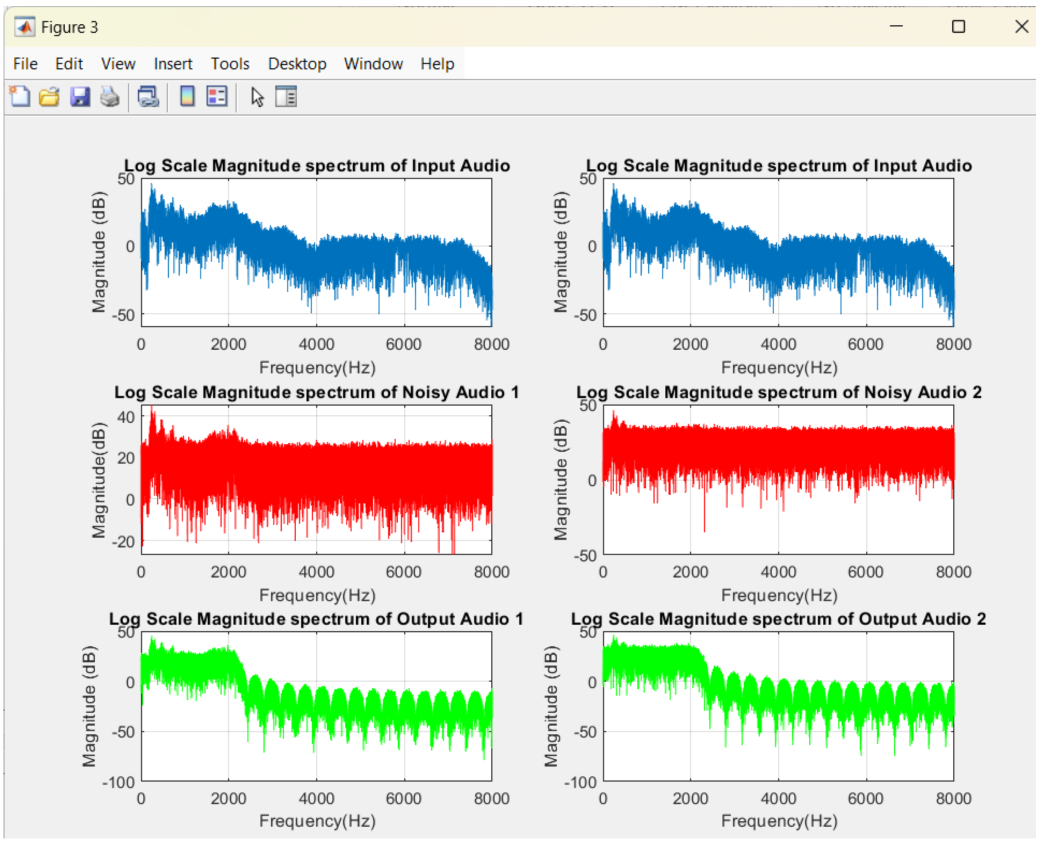


Fig. 6.7: Log scale Magnitude spectrum of Input, Noisy signals, and output signals

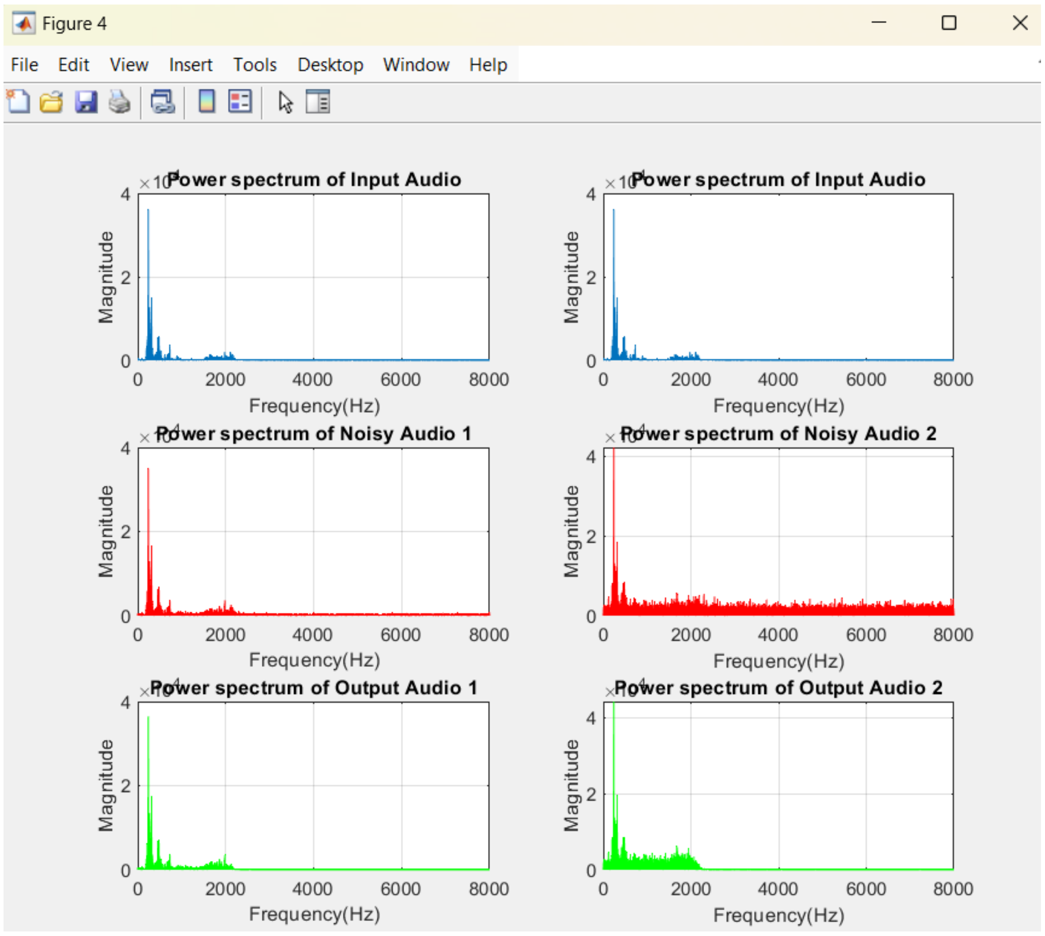
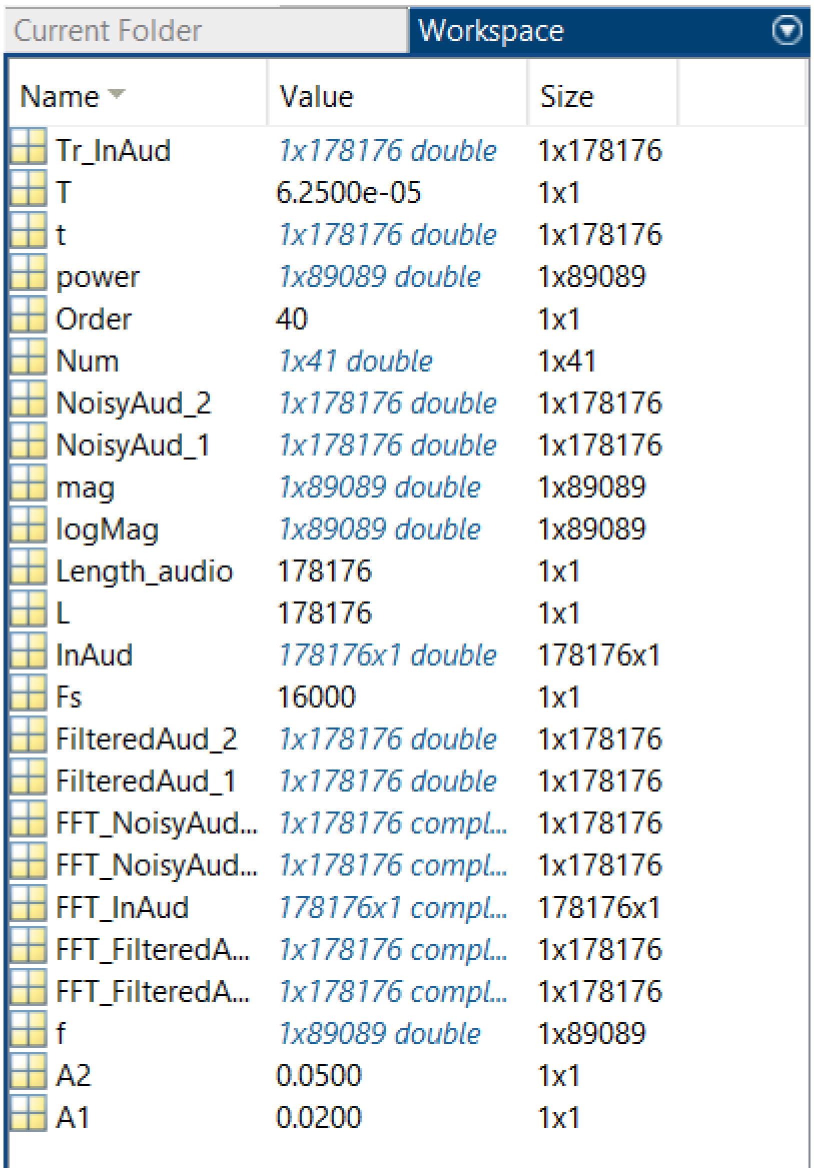


Fig. 6.8: Power spectrum of Input, two noisy signals, and the respective output signals after applying the filter.



| Name | Value | Size |
|---|---|---|
| Tr_InAud | 1x178176 double | 1x178176 |
| T | 6.2500e-05 | 1x1 |
| t | 1x178176 double | 1x178176 |
| power | 1x89089 double | 1x89089 |
| Order | 40 | 1x1 |
| Num | 1x41 double | 1x41 |
| NoisyAud_2 | 1x178176 double | 1x178176 |
| NoisyAud_1 | 1x178176 double | 1x178176 |
| mag | 1x89089 double | 1x89089 |
| logMag | 1x89089 double | 1x89089 |
| Length_audio | 178176 | 1x1 |
| L | 178176 | 1x1 |
| InAud | 178176x1 double | 178176x1 |
| Fs | 16000 | 1x1 |
| FilteredAud_2 | 1x178176 double | 1x178176 |
| FilteredAud_1 | 1x178176 double | 1x178176 |
| FFT_NoisyAud... | 1x178176 compl... | 1x178176 |
| FFT_NoisyAud... | 1x178176 compl... | 1x178176 |
| FFT_InAud | 178176x1 compl... | 178176x1 |
| FFT_FilteredA... | 1x178176 compl... | 1x178176 |
| FFT_FilteredA... | 1x178176 compl... | 1x178176 |
| f | 1x89089 double | 1x89089 |
| A2 | 0.0500 | 1x1 |
| A1 | 0.0200 | 1x1 |

Fig. 6.9: Workspace after executing the program in MATLAB

## 7. Conclusion:

In this, a Low pass filter has been designed which stops the high frequencies after the given cutoff frequency. A low pass filter has been designed based on its application of removal of noise. The undesirable frequencies that are above the cutoff frequency are eliminated with the use of a low pass filter. The noisy audio file is used as the input to the filter. After setting the cutoff frequency of the FIR Kaiser window of the low pass filter, the filtered audio file with less noise compared to noisy audio has been obtained. Hence the designed filter has removed the high-frequency audio which can be even recognized as output audio filtered.

We have included random noises with two varying amplitudes. By experimenting we found that a lower-order filter can remove the noise to a lesser extent. However, when we increase the amplitude of the noise, the designed filter is unable to remove the noise efficiently. Hence, the low pass filter with a higher order is required.

In summary, applying a low-pass filter to an audio file to remove random noise is a strategic and essential process. It aligns with the broader goal of enhancing the quality, clarity, and intelligibility of audio signals across various applications and environments. One has to appropriately balance the trade-off between increasing the order of the filter and maintaining its efficiency.

## 8. Applications:

Applying a low-pass filter to an audio file to remove random noise finds application in various scenarios where the goal is to improve the quality, clarity, and intelligibility of the audio signal by attenuating or eliminating unwanted high-frequency noise. Here are specific applications of this noise reduction technique:

1. Audio Forensics:

In forensic audio analysis, where clear identification of voices or sounds is crucial, a low-pass filter can be used to remove unwanted high-frequency noise and isolate the essential components of the audio signal.

2. Radio Broadcasting:

In radio broadcasting, especially in AM (Amplitude Modulation) systems, atmospheric interference or electrical noise can affect the quality of the broadcast. Using a low-pass filter helps in removing unwanted high-frequency noise, ensuring a cleaner transmission.

3. Medical Audio Recordings:

In medical applications where audio recordings are used for diagnostics or monitoring, applying a low-pass filter can help reduce electron interference or high-frequency noise, ensuring that critical information in the lower frequency range is preserved.

In essence, applying a low-pass filter to an audio file to remove random noise is a versatile and widely used technique with applications spanning communication, entertainment, scientific research, and various other fields where high-quality audio is essential.

# 9. Limitations and Future Scope:

## 9.1. Limitations:

1. High Computational Requirements: FIR filters can be computationally demanding, especially with long filter lengths, which may limit their use in applications with strict computational constraints.

2. Non-Adaptability in Real-Time: FIR filters are not easily adaptable to real-time changes in filter specifications, making them less suitable for applications that require dynamic adjustments or adaptive filtering in real-time scenarios.

## 9.2. Future Scope:

The future scope of Low-Pass Filters (LPF) in digital signal processing holds significant promise as technology continues to advance. With the growing ubiquity of digital systems and the increasing complexity of signal-processing applications, LPFs are expected to play a crucial role in enhancing the efficiency and reliability of various technologies. In fields like telecommunications, LPFs will continue to be integral for anti-aliasing and ensuring the integrity of transmitted signals. Moreover, as the Internet of Things (IoT) expands, LPFs will contribute to noise reduction and signal conditioning in sensor networks, improving the accuracy of data collected from various devices. In emerging technologies like 5G communication, LPFs will be essential for managing signal quality and mitigating interference. Additionally, LPFs are likely to find applications in advanced medical devices, audio processing systems, and artificial intelligence, where the precise filtering of signals is essential for optimal performance. As DSP capabilities evolve, the future of LPFs involves innovative designs, adaptive filtering techniques, and integration with other advanced signal processing algorithms to address the dynamic and diverse needs of modern applications.

## 10. References:

1. https://en.wikibooks.org/wiki/Digital_Signal_Processing/Windowing
2. https://www.dsprelated.com/freebooks/sasp/Kaiser_Window.html
3. https://www.mathworks.com/discovery/filter-design.html
4.https://in.mathworks.com/matlabcentral/answers/217711-how-to-addrandom-noise-to-a-signal
5. https://in.mathworks.com/help/signal/ug/fir-filter-design.html
6. https://in.mathworks.com/help/signal/gs/designing-the-filter.html